Analysis of Algorithms | **Notes on Matchings**
CS 6820 Fall 2014 | **September 8-12**

Given a graph $G = (V, E)$ with nodes V and edges E, a matching M is a subset of edges $M \subset E$ such that each node has degree at most 1 in M. A node v is matched in M if it has an adjacent edge. A matching M is a perfect matching if all nodes are matched. Both of the recommended text books have discussions on the matching problem, its applicants, so we won't repeat this here. In this note we will discuss two topics. First cover the greedy algorithm for max weight matching, and the the Hopcroft -Karp $O(\sqrt{|V|}|E|)$ algorithm for finding a maximum matching (with no weights).

Greedy Algorithm

Given a graph and weights $w_e \geq 0$ for the edges, the goal is to find a matching of large weight. The greedy algorithm starts by sorting the edges by weight, and then adds edges to the matching in this order as long as the set of a matching. So a bit more formally:

Greedy Algorithms for Matching

```
$M = \emptyset$
For all $e \in E$ in decreasing order of $w_e$
   add $e$ to $M$ if it forms a matching
```

The greedy algorithm clearly doesn't find the optimal solution. To see an example, consider a path of length 3 with two edges of weight 1, and the middle edge of weight $1 + \epsilon$. The greedy algorithm results in a single edge matching of weight $1 + \epsilon$, while the optimum is the two edge matching of weight 2. Essentially a factor of 2 off. We claim that this example is worst possible

Theorem 1. *The weight of the matching M returned by the greedy algorithm is at least half of the weight of any matching M^*.*

Proof. Let M^* is a matching of maximum weight, and M be the matching returned by the greedy algorithm. Note that for any edge $e \in M^* \setminus M$, there is a reason e didn't get into the greedy matching M, a previously considered edge, lets call it $f(e)$ that has higher weight, and shares an end-node with e. If there are multiple such edges, let $f(e)$ be either of the two such edges.

- Note that $w_{f(e)} \geq w_e$ as we add edges in greedy order.
- For an edge f there can be $f(e)$ for up to at most two edges e, conflicting with edge f at the two different ends.

Putting these two facts together, we get the following inequalities

$$\sum_{ein \in M^*} w_e \geq \sum_{e \in \in M^*} w_{f(e)} \geq \sum_{f \in \in M} 2w_f$$

proving the theorem. □

Maximum Matching and Augmenting Path

Next consider finding the maximum size matching without weights, so just wanting to maximum $|M|$ for a matching M. The main tool for finding maximum size matching is augmenting path. Given a matching M in a graph G, we say that a path P or cycle C is alternating if it it alternates between matched and unmatched edges. If such an alternating path starts and ends at an unmatched node, its called an *augmenting path*.

In proving things about matching, we'll use the notion of symmetric difference. For two sets A and B we say $A \triangle B = (A \cup B) \setminus (A \cap B)$, the set of nodes contained in exactly one of the two sets A and B.

Using this it is not hard to see that an alternating path P, can be used to find a matching of one larger size, namely $M' = M \triangle P$ is a larger matching. Maybe more surprisingly, Berge proved that if M is not of maximum size, there is an augmenting path.

Lemma 2. *Given a matching M is of maximum size if and only if M has no augmenting path P.*

Proof. For an augmenting path P the set $M' = M \triangle P$ is a matching of one larger size.

To see the opposite, let M^* be a matching larger size than M, and consider $M^* \triangle M$. Note that in this set all nodes have degree at most 2 (one from M and one from M^*), so it consist of a set of disjoint paths and cycles. Further, these paths and cycles have to alternate edges from M and M^* so they are alternating paths and cycles for matching M. An alternating cycle must have the same number of M and M^* edges. Since M^* is larger, some path P must have more M^* edges. The only way a path can have more M^* edges, if its a path starting and ending with an M^* edge, at an nodes unmatched by M, so its an augmenting path. □

To be used later, we include here a stronger version of this lemma that we'll need:

Lemma 3. *Given a matching M so that there is a matching M^* that has k more edges (that is, $|M^*| \geq |M| + k$) for some $k > 0$, then M has an augmenting path of length at most $|V|/k - 1$ (using at most $|V|/k$ nodes).*

Proof. Recall the proof above thinking about $M^* \triangle M$. Since M^* is k larger, the symmetric difference must have at least k paths with more M^* edges than M edges. These are k disjoint augmenting paths, so one of them must have at most $|V|/k$ nodes. □

Maximum Matching Algorithm

Using Lemma 2 above, we can find maximum matching, if we can find augmenting path. Finding augmenting path is hard in general graphs, but it is easier in bipartite graphs. The following directed graph constriction works well.

For a bipartite graph $G = (V, E)$ and a matching M in G we call consider the following *residual graph* R_M with nodes V and two additional nodes s and t, and edges as follows, using A and B to denote the two sized of the bipartite graph.

- For each unmatched node $a \in A$ add edges (s, a) to R_M
- For each unmatched node $b \in B$ add edges (b, t) to R_M
- For each edge $(a, b) \in M$ add directed edge (b, a) to R_M
- For each edge $(a, b) \notin M$ add directed edge (a, b) to R_M

So edges in the matching are directed B to A, while edges not in the matching are directed A to B. Any path in R_M must alternate going A to B to A, etc. A path from s to t of length 3 is just one unmatched edges (with nodes s and t added). A path from s to t of length 3 consist of s, an augmenting path of length 3 and then node t. More generally, we get the following lemma.

Lemma 4. *Paths in R_M from s to t are in one to one correspondence to augmenting paths in G for matching M.*

Using a simple DFS or BFS algorithm to find path in R_M we get an $O(|V||E|)$ maximum matching algorithm as follows.

Maximum Matching in Bipartite Graphs

```
$M = \emptyset$
While path is found
   Construct directed graph $R_M$
   find a path $P$ from $s$ to $t$ in $R_M$
   Augment $M$ using path $P$.
```

Theorem 5. *The above algorithm finds a maximum matching, in $O(|V|((|E| + |V|)$ time.*

The Hopcroft-Karp algorithm finds a maximum matching in the improved $O(\sqrt{|V|}|E|)$ time. To do this we need to focus only on the shortest augmenting paths, and find many of them in one iteration. First, to focus on the shortest path, after constructing R_M we delete all edges that are not in shortest path from s to t. Lets call this graph R'_M. Next we will want to find a maximal set of disjoint path in R'_M, and then augmenting using all of them at once. Here is the resulting algorithm.

Hopcroft-Karp maximum Matching in Bipartite Graphs

```
$M = \emptyset$
```

```
While path is found
   Construct directed graph $R'_M$
   $\mathcal{P} = \emptyset$ /* the set of augmenting path found*/
   While additional paths are found
      find a path $P$ from $s$ to $t$ in $R'_M$ disjoint from all the paths in $\mathcal{P}$
   Augment $M$ using all the path $\mathcal{P}$.
```

First focus on the inner loop, aiming to find a maximal set of disjoint shortest paths in a graph efficiently. The very first of these inner loops, when $M = \emptyset$, the algorithm is the greedy algorithm discussed above. As we have no weights (or all weights are 1), any ordering of the edges will do, and the matching we find is at least 1/2 of maximum size matching. It is not hard to see that this greedy algorithm (without the sort of the edges) can be implemented in linear time ($O(|E| + |V|)$ time).

Next we show that this is also true in any later iteration. Note that all paths in R'_M will be shortest, so the inner loop doesn't need to focus on finding short paths, only on finding disjoint paths. We suggest to implement this via Depth First Search (DFS). DFS finds a path from s to t. After it found some number of paths, we start again from s in looking for additional paths, but do not have to search again the edges the we searched previously.

Lemma 6. *In any directed graph $G = (V, E)$ with nodes s and t we can find in $O(|E| + |V|)$ time a set of path $\mathcal{P}$ from s to t in G that are disjoint (expect for s and t), and no further s to t path in G is disjoint from all of the paths in $\mathcal{P}$.*

This implements the inner `while` loop in linear time. Next we have to limit the number of iterations of the outside `while` loop. This will come in two steps. First we show that the length of the shortest path increases between every two iterations.

Lemma 7. *For any matching M in a bipartite graph $G = (V, E)$ and a maximal set of disjoint shortest paths $\mathcal{P}$ in R_M and let d be the length of the shortest path in R_M. Now consider the matching M' obtained by augmenting M by all the augmenting paths in $\mathcal{P}$. The shortest path in $R_{M'}$ has length at least $d + 2$.*

Proof. In the directed graph R_M let $d(v)$ denote distance from s to v in this graph, so $d(t) = d$. Note that any shortest path will only use edges that go from a node v at some distance $d(v)$ to a node at distance $d(v) + 1$. Now consider the way R_M and $R_{M'}$ differ. To get $R_{M'}$ we delete some edges from R_M: the edges leasing s and the edges entering t that are along the paths in $\mathcal{P}$. For the other edges along the paths in $\mathcal{P}$ each edge changes its direction (as it switched between being in the matching and outside the matching). If an edge along a path went from a node v at some distance $d(v)$ to a node w at distance $d(v)+1$, the edge goes from distance d(v)+1 to $d(v)$ (according to the distances in R_M). So no edges are added that can make distanced shorter. Further, $\mathcal{P}$ is already a maximal size, so no shortest path can be disjoint from $\mathcal{P}$. This shows that the distance $d' > d$. To see that it is at least 2 bigger, note that due to the bipartite graph we start with, the distance has to be odd. □

To show that the number of iterations is low, we put together Lemmas 6 and 3. By Lemma 6 it will take at most $\sqrt{|V|}/2$ iteration of the outer loop to reach a matching M where the shortest path length in R_M is at least $\sqrt{|V|}$. We claim that at this point the size of the matching is almost optimal, more concretely that $|M^*| - |M| \leq \sqrt{|V|}$. To see why, assume that $|M^*| - |M| = k$, and use Lemma 3 to see that the shortest augmenting path now must have length $d \leq |V|/k$. So $d \geq \sqrt{|V|}$ implies that $k \leq \sqrt{|V|}$. Each iteration of the outer loop adds at least one edge to the matching, so we get our main result.

Theorem 8 (Hopcroft-Karp)**.** *The Hopcroft-Karp algorithm finds a maximum matching, has at most $O(\sqrt{|V|})$ iterations of its outer loop, and can be implemented in $O(\sqrt{|V|}((|E|+|V|)$ time.*